WHITE PAPER

Creating Flexible Hardware Systems with FPGA Partial Reconfiguration

Nicolas Morin,
FPGA Design Engineer

abaco SYSTEMS

Creating Flexible Hardware Systems with FPGA Partial Reconfiguration

Background

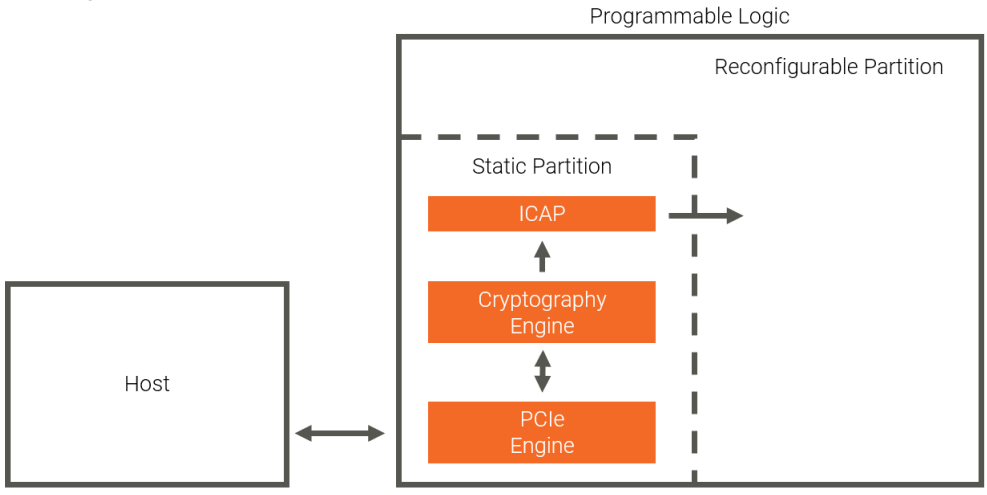
Partial Reconfiguration (PR) allows FPGAs to dynamically change modules without disrupting other parts of the design. This is a feature that FPGA vendors are building into their newer generations of FPGAs, allowing for increased flexibility and functionality in digital systems. Users can partition the FPGA fabric into reconfigurable regions which are then reprogrammed using partial configuration files. PR proves beneficial in systems that communicate through PCIe™, which allows a user to dynamically reload a subset of the FPGA image without losing PCIe communication. It also provides a critical method of Intellectual Property (IP) protection as it removes the need to store sensitive data in non-volatile memory on the FPGA carrier.

In the new Xilinx® RFSoC technology, PCIe PR through the Programmable Logic (PL) requires special considerations throughout the design process. This paper discusses the use cases of partial reconfiguration as well as considerations when designing partial reconfiguration firmware using the Xilinx Vivado design tool targeting the RFSoC.

Use Cases

Bitstream Encryption

Security has become critically important in embedded aerospace and defense systems. Encryption is heavily used for assuring data integrity and privacy of communication systems. Dynamic partial reconfiguration allows for a secure method of protecting FPGA configuration files, with most of the design living in the reconfigurable partition. The static region of the device would contain an external interface, a cryptography engine, and a path to the internal configuration access port (ICAP) leaving the rest of the FPGA's resources free to be reprogrammed.

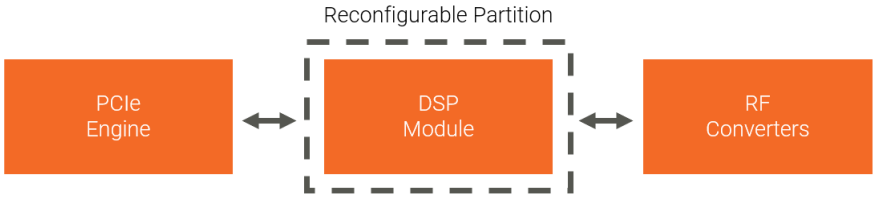


After the static configuration is loaded, the cryptography engine creates a public-private key pair. The FPGA then sends the public key to a host computer that holds the partial bitstream. The host then uses the public key to encrypt the partial bitstream and send it to the cryptography engine for decryption. The cleartext partial bitstream is then sent to the ICAP to configure the remainder of the FPGA without fear of interception.

Creating Flexible Hardware Systems with FPGA Partial Reconfiguration

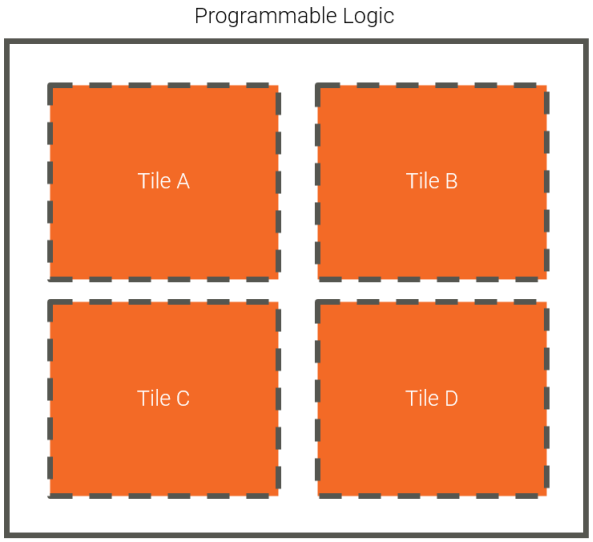
Dynamically Reconfigurable DSP Algorithms and SDR

Software defined radio enables flexible radio architectures that adapt to various protocols, allowing for the customization of bandwidths, modulation frequencies, and decimation rates on the fly. Using the RFSoC, the flexibility of software defined radio is enhanced using hardware acceleration in the programmable logic. It's necessary to further process the digital signals in hardware, and in different ways without losing communication with a host device. This can be solved by making FIR filters, FFTs, correlators, equalizers, encoders and pattern recognition logic dynamically reconfigurable.



Fault-Tolerant System

Advances in technology are enabling higher levels of automation in military systems. One example of this is the ability to detect faults which occur from aging and environmental factors. Subsystems of the FPGA cannot be disrupted while subsystems are redesigned, especially in mission-critical environments. Dynamic partial reconfiguration provides a solution by leaving subsystems static, while reducing reconfiguration time to correct the fault. This can be accomplished by partitioning the FPGA into tiles. Tiles with faults can be reprogrammed with a partial bitstream in a fraction of the time needed to program the entire FPGA, all while maintaining functionality of the other tiles.



Creating Flexible Hardware Systems with FPGA Partial Reconfiguration

Potential solutions for dynamically loading the FPGA image on RFSoC

PCIe through the PL

The RFSoC can reconfigure the PL from inside the PL itself through the internal configuration access port (ICAP). This interface supports a 32-bit wide bitstream data at 200 MHz, giving a bandwidth of 800 MB/s. While some UltraScale+ devices offer a dedicated link to the ICAP through a specific PCIe block (called the MCAP), the RFSoC does not have this feature. When designing for partial reconfiguration from inside the PL, the ICAP instantiation along with the necessary partial bitstream data source and accompanying data path elements should be implemented in the static partition.

PCIe or Ethernet through the PS

Partial reconfiguration can also be achieved through the Processor Subsystem (PS) of the RFSoC. The PL can be reconfigured through the PS via the processor configuration access port (PCAP). Since the PCAP is not accessible from the PL, a design would need to support transporting the bitstream over PCIe or Ethernet to Linux® running on APU. Once transported, the bitstream is sent to the fpga_manager kernel driver which invokes the necessary secure calls to the CSU to DMA the bitstream to the PCAP.

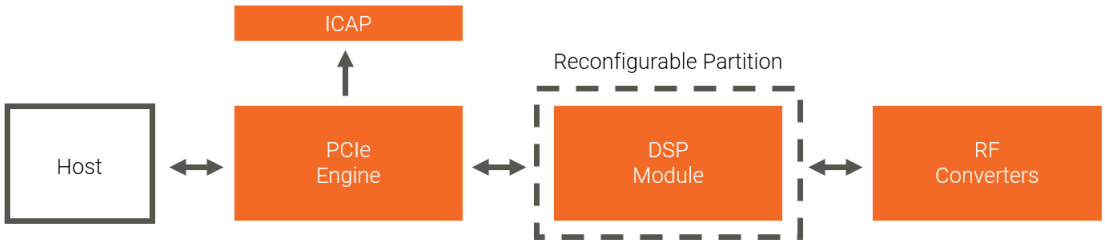
Optimal Solution

The optimal solution depends on the RFSoC carrier design and the system design for the end application. If the system is designed to control the RFSoC carrier through PCIe in the PL, then this article describes how to be successful with that approach.

PCIe Partial Reconfiguration – Abaco BSP example

In this example, we will walk through the process of converting an existing RFSoC PL design based in Vivado IP Integrator into a partial reconfiguration project. The project is based on **Xilinx's XAPP1338**, which provides a method to partially reconfigure through a PCI Express™ interface. We wish to design the following system:

Creating Flexible Hardware Systems with FPGA Partial Reconfiguration



Several steps must take place in order to turn an existing design into a partially reconfigurable one:

1) We must convert reconfigurable modules into a format supported by a partial reconfiguration project in Vivado. Reconfigurable modules defined inside block diagrams are **currently not supported by the tool**. It is possible to work around this, however, if it is necessary to build reconfigurable modules in your design through IP Integrator.

2) The design must provide a route to a **supported configuration access port**. We will be using the ICAP in this example.

3) We must also decouple the logic between the static and reconfigurable partitions to prevent corruption during reconfiguration. This will be accomplished using the **Partial Reconfiguration Decoupler IP from Xilinx**.

4) We will **floorplan the reconfigurable partition** by inserting Pblocks.

5) We will then **format the bitstream** using the write_cfgmem command for use by the ICAP.

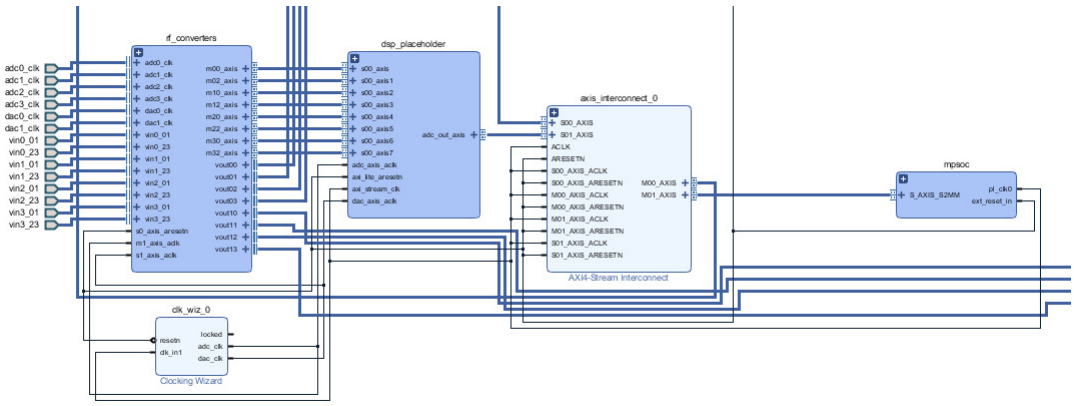
6) We will walk through the process of supporting partial reconfiguration by the ICAP using PetaLinux.

The following workflow assumes that you are starting with an already completed block design in Vivado 2018.3 or newer, targeting the Xilinx RFSoC. If using PetaLinux, the paper also assumes the use of PetaLinux 2018.3 or newer.

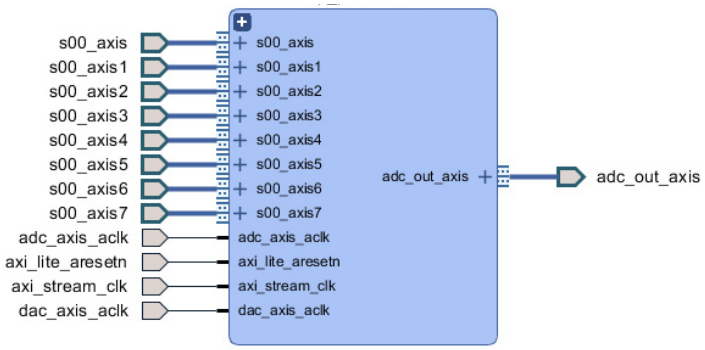
Packaging Reconfigurable Modules as IP

Since IP source files are allowed inside reconfigurable modules, we will package a specific block as an IP and import back into the project. We will then make necessary connections from the original block design to the reconfigurable partition through HDL code. As an example, we will start with the following design.

Creating Flexible Hardware Systems with FPGA Partial Reconfiguration

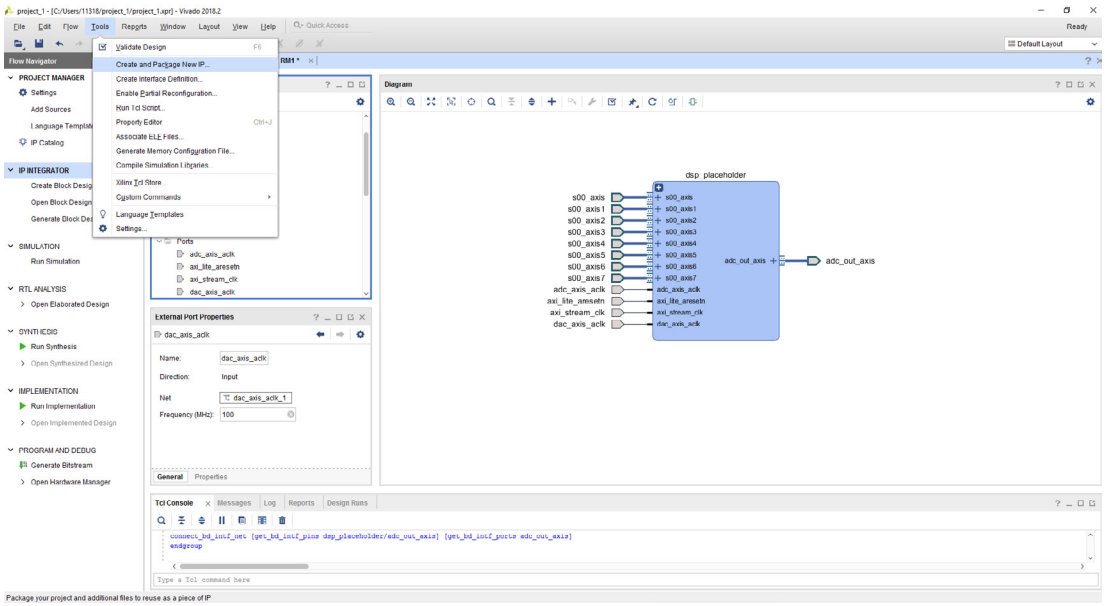


We wish to make the dsp_placeholder block reconfigurable. First, copy the dsp_placeholder block and paste it into a new block design. We are going to title the block design RM1. Create interface ports to the block as shown in figure.

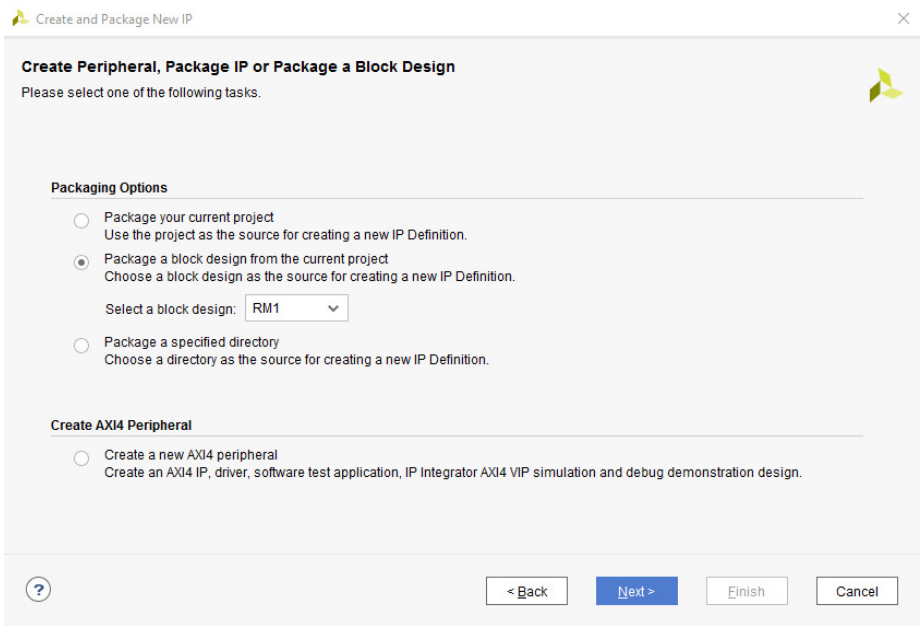


Make sure all interface ports have the correct associated clocks and reset polarity. If using an AXI memory mapped interface, make sure the address editor is configured correctly. Next, navigate to Tools > Create and Package New IP.

Creating Flexible Hardware Systems with FPGA Partial Reconfiguration



Select Package a block design from the current project. Select RM1.

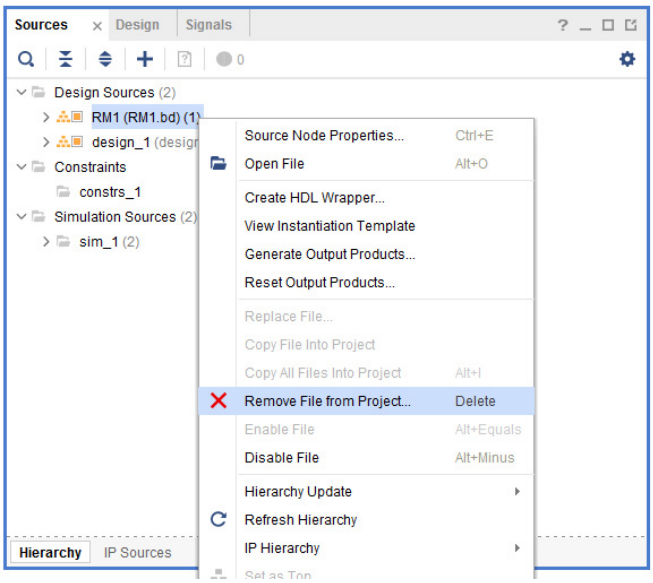


Choose a User IP directory corresponding to the current project. If you do not have one, then create one. We will save the IP inside the location <current_project>/IP/RM2.

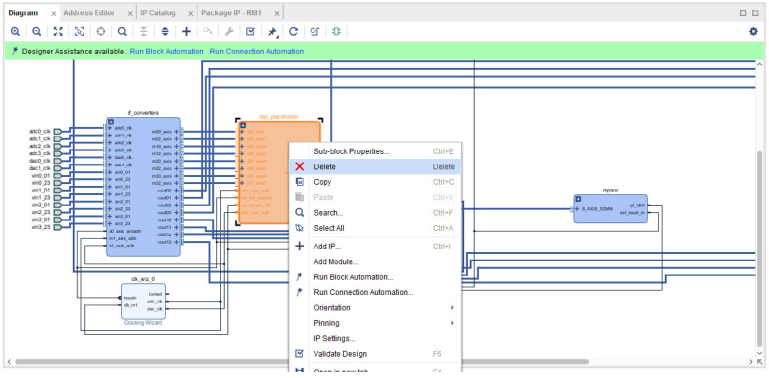
Creating Flexible Hardware Systems with FPGA Partial Reconfiguration

Instantiating the Reconfigurable Module

We will now instantiate the RM1 IP outside the block diagram. We will make all connections from the RM1 IP to the block diagram with HDL code. We can now remove the RM1.bd file from the source hierarchy.

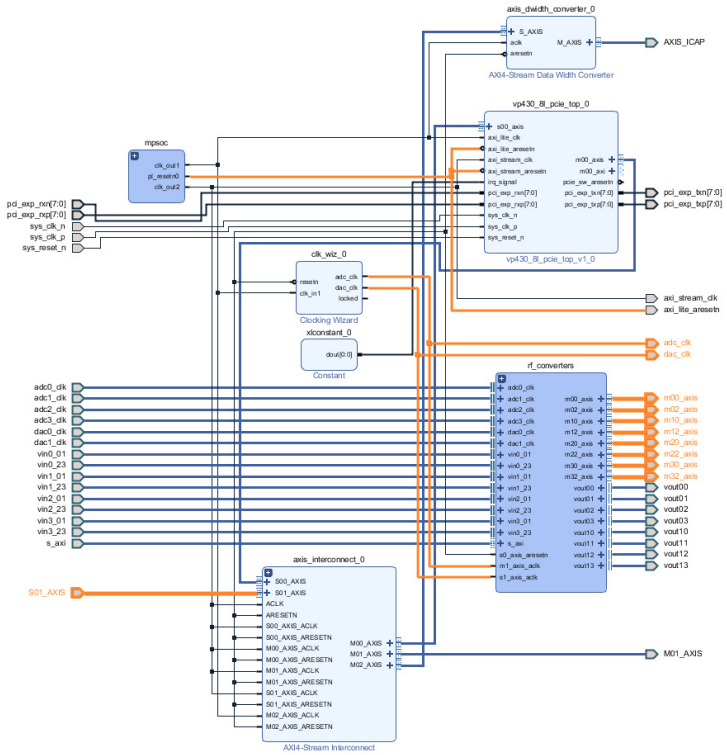


We now need to delete the dsp_placeholder block from the original block design.

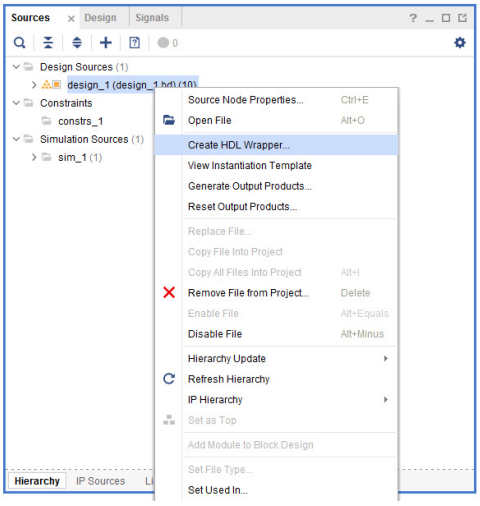


We will now have disconnected ports. Create interface ports to route them off the block diagram. Make sure all ports have the correct associated clock ports and all reset signals have the correct polarity.

Creating Flexible Hardware Systems with FPGA Partial Reconfiguration

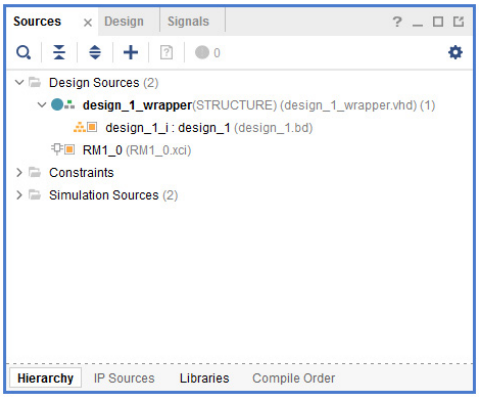


Right click on the block diagram source and create an HDL wrapper. Allow user edits to the wrapper file.

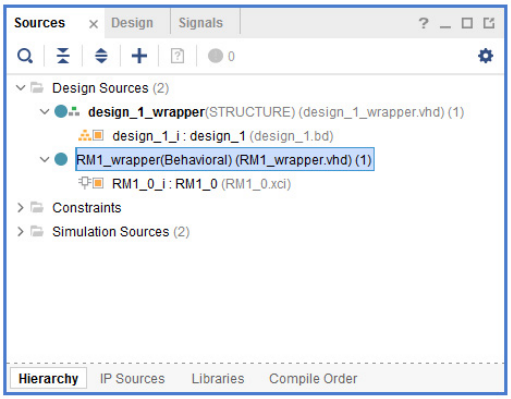


Import the new RM1 IP. It should currently exist at the same level as the block design wrapper.

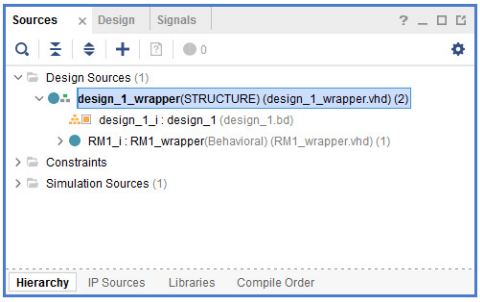
Creating Flexible Hardware Systems with FPGA Partial Reconfiguration



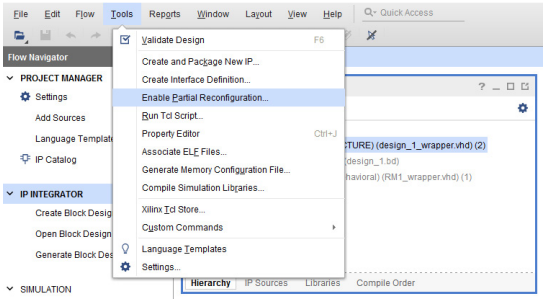
Create a wrapper for the reconfigurable module. The instantiation template can be found in the IP sources pane.



Instantiate the reconfigurable module wrapper underneath the block design wrapper, so it sits one level below the top level.

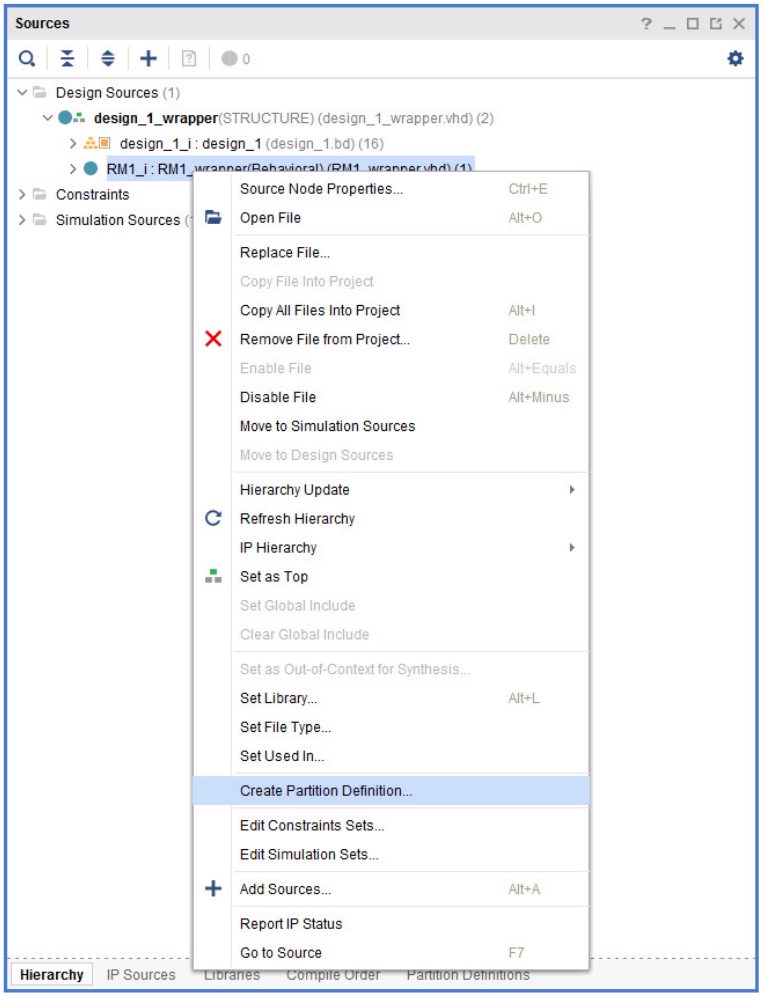


We are now ready to enable the partial reconfiguration mode. Go to Tools > Enable Partial Reconfiguration.



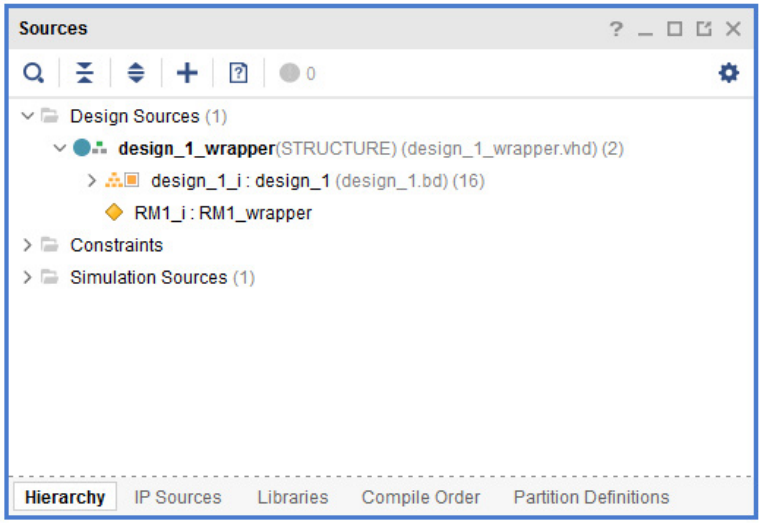
We now have the option to create Partition Definitions. We also have a new Partial Reconfiguration Wizard section inside the flow navigator. Right click on the reconfigurable module and select Create Partition Definition.

Creating Flexible Hardware Systems with FPGA Partial Reconfiguration



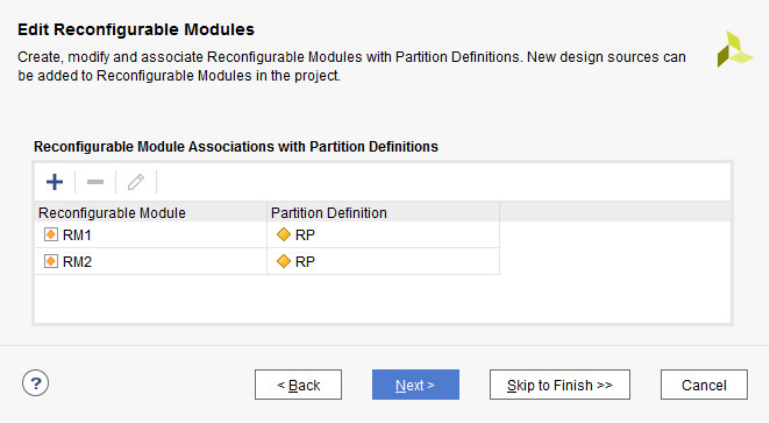
We are going to title the partition RP. The reconfigurable module name will be called RM1. You will then notice that RM1 inside the source hierarchy has a yellow diamond next to it to signify it as a Reconfigurable Partition.

Creating Flexible Hardware Systems with FPGA Partial Reconfiguration



Adding additional Reconfigurable Modules

To add another Reconfigurable Module, click on the Partial Reconfiguration Wizard. Add another design source. We will title it RM2.

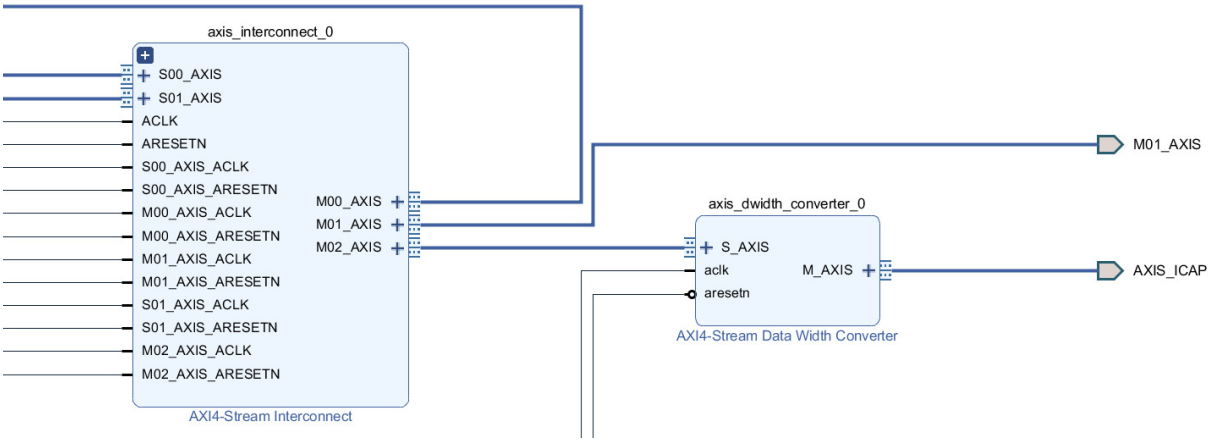


Automatically create configurations as well as configuration runs. You should now see two reconfigurable modules listed under the Partition Definitions pane of Vivado.

Creating Flexible Hardware Systems with FPGA Partial Reconfiguration

Instantiating the ICAP

In this design, we will be partially reconfiguring through the internal configuration access port (ICAP). This SelectMAP style interface exists inside the PL of the RFSoC. We will make a path that routes from the PCIe logic block through an AXI-Stream Interconnect, and a data width converter to the ICAP. The AXI Stream Interconnect will allow the ICAP to be clocked differently than the rest of the design. The interconnect normally provides data width conversion to create a width of 32 bits for the ICAP; however, we have found it to be more reliable to instantiate a data width converter deliberately when routing an interface off the block diagram.



Creating Flexible Hardware Systems with FPGA Partial Reconfiguration

The maximum clock speed that the RFSoC ICAP supports is 200 MHz. The ICAP's bit width is 32 bits on the RFSoC. We will now need to instantiate the ICAP inside the block design wrapper. We can do this with the following code:

```
ICAPE3_inst : ICAPE3
  generic map (
    ICAP_AUTO_SWITCH => "DISABLE"
  )
  port map (
    AVAIL => axis_icap_tready,
  O => open,
  PRDONE => prdone,
  PRERROR => prerror,
  CLK => axi_lite_clk,
  CSIB => axis_icap_tvalid_not,
  I => axis_icap_tdata,
  RDWRB => '0'
);
```

The port description for the ICAPE3 are listed below:

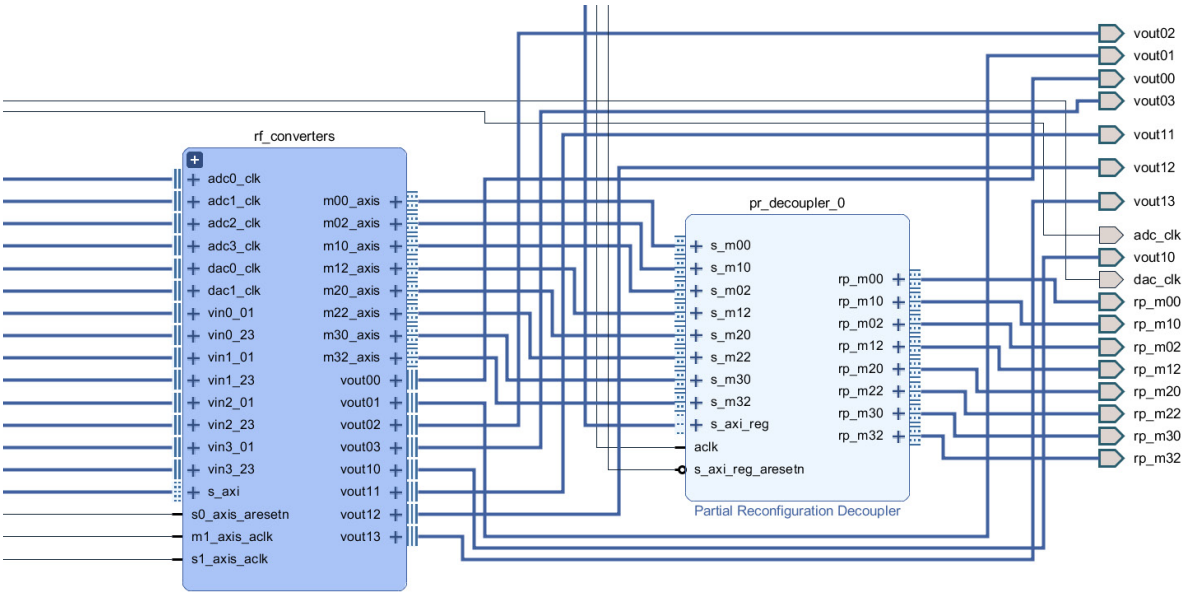
Avail	Output	1	High when ICAP is available
CLK	Input	1	Clock input
CSIB	Input	1	Active-low ICAP enable
I	Input	32	Configuration data input bus
O	Output	32	Configuration data output bus
PRDONE	Output	1	High when partial reconfiguration is complete. Goes low when FDRI packet is seen and goes back high when DESYNC is seen, or EOS is high
PRERROR	Output	1	High when partial reconfiguration error is detected
RDWRB	Input	1	Read (high) or write (low) select input

More information regarding the ICAPE3 primitive can be found in Xilinx's **UG974**.

Creating Flexible Hardware Systems with FPGA Partial Reconfiguration

Partial Reconfiguration Decouplers

Depending on the design, the use of Partial Reconfiguration Decouplers may be required. The reconfigurable partition can cause erroneous writes to the static logic while the PL is reconfiguring. Partial Reconfiguration Decouplers act as a set of multiplexers. We will control the decoupler with an AXI memory-mapped interface, although it is possible to control it with a signal or an AXI stream interface. We will instantiate a Partial Reconfiguration Decoupler between the rf converters and the reconfigurable modules as shown:

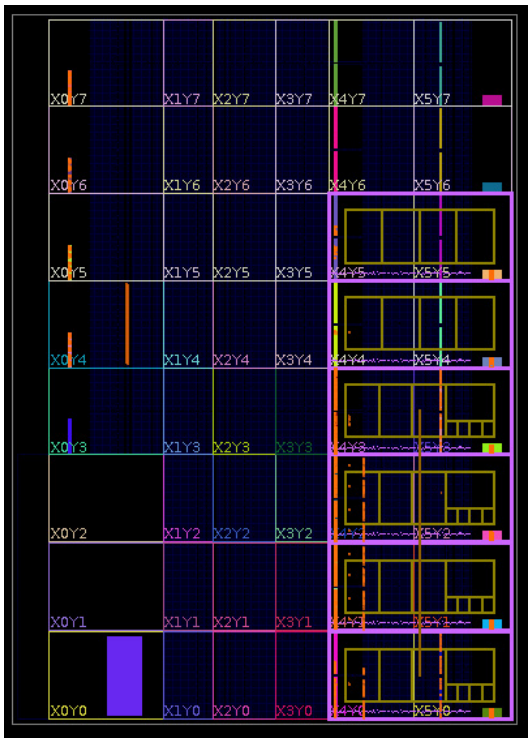


Assign the decoupler to an address in the register map. Writing a value of 0x1 to this address will decouple the logic, while writing a 0x0 will recouple the logic. More information can be found in **Xilinx's PG227**.

Creating Flexible Hardware Systems with FPGA Partial Reconfiguration

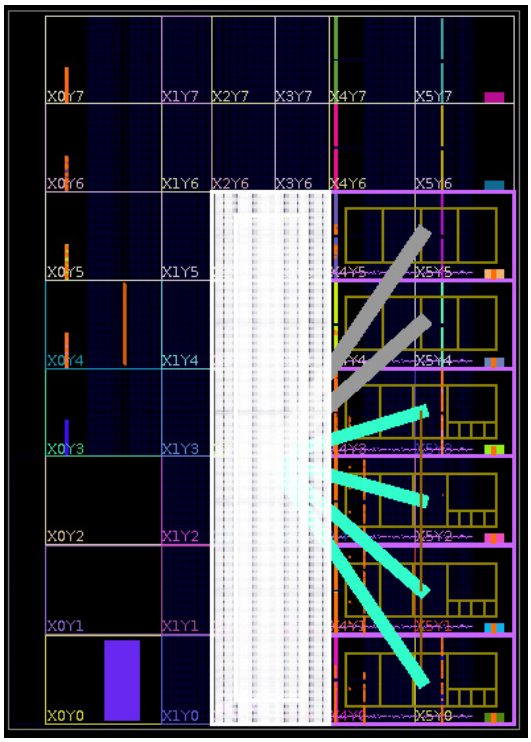
Floorplanning

After synthesis, we will need to floorplan the design. Each Reconfigurable Partition is required to have a Pblock associated with it to assign physical resources to Reconfigurable Modules as well as physically constrain the partition in a location on the chip. On the RFSoC, you will notice that the RF converters have Pblocks associate with them by default in the bottom right. More information can be found on Floorplanning can be found in **Xilinx's UG909**.



We will place a Pblock next to the RF converters. This will help meet timing requirements, since the DSP module physically connects to them. Create a Pblock by right clicking on RM1 in the netlist pane on the left. Select Floorplanning > Draw Pblock. Click and draw on the chip where you would like to create the Pblock. We will draw the following block:

Creating Flexible Hardware Systems with FPGA Partial Reconfiguration



Notice the cyan and grey rectangles connecting the Pblocks. This is to provide visual aid as to which floorplanned modules are physically connected. After the Pblocks are drawn, the design is then ready for implementation and bitstream generation.

Formatting the Bitstream

We will need to format the bitstream for use by the ICAP. First, generate a bit file as you would normally. You can then use the following command inside the Vivado TCL console:

```
write_cfgmem -format BIN -size 128 -interface SMAPx32 -loadbit {up
0x00000000 "/path_to_partial_bitstream.bit"} -file /path_to_partial_
bitstream.bin
```

More information about bitstream formatting can be found in Xilinx's **UG909**.

Software Implications

The default configuration access port available for partial reconfiguration on the RFSoC is the PCAP. This is controlled by the **pcap_ctrl register inside the CSU**. We must relieve control from the PCAP and set it to the ICAP by setting bit 0 of address 0xFFCA3008.

Creating Flexible Hardware Systems with FPGA Partial Reconfiguration

FIELD NAME	BIT	TYPE	RESET VALUE	DESCRIPTION
Pcap_pr	0	Rw	0x1	Controls the method for PL partial reconfiguration 0x0 – ICAP/MCAP 0x1 – PCAP

If using PetaLinux, access to the CSU registers is turned off in the default configuration mode. In order to gain access, we need to set the SECURE_ACCESS_VAL flag inside the xpfw_config.h file in the PMUFW. After building the image with the correct firmware, we can then send the following command to the PetaLinux console:

```
# echo 0xFFCA3008 0xFFFFFFFF 0x00000000 > /sys/firmware/zynqmp/config_reg
```

Additionally, any software interacting with devices in the PL must be aware of when partial reconfiguration is taking place. An MMIO access may fail and potentially cause the APU to lock up. Complex interactions with devices may be interrupted and leave software in a bad state. Due to the nature of partial reconfiguration, users must take care to notify software before and after a partial reconfiguration event so that proper shutdown and/or reinitialization can take place.

Conclusion

There are many examples of when PCIe PR through the PL on RFSoC-based carriers is the optimal solution. It can be difficult to accomplish this without the proper knowledge and experience up front. The example provided in this white paper offers the best possible starting point for achieving success in this type of application.

Xilinx and Ultrascale+ are trademarks of Xilinx, Inc. PCIe and PCI Express are trademarks of PCI-SIG. Linux is the registered trademark of Linus Torvalds. All other trademarks are the property of their respective owners.